Evolving Network Architectures with Custom Computers for Multi-Spectral Feature Identification

Reid Porter, Maya Gokhale Neal Harvey, Simon Perkins and Cody Young
NIS-2, Space and Remote Sensing Sciences
Los Alamos National Laboratory, NM, 87544
rporter@lanl.gov

Abstract

This paper investigates the design of evolvable FPGA circuits where the design space is severely constrained to an interconnected network of meaningful high-level operators. The specific design domain is image processing, especially pattern recognition in remotely sensed images. Preliminary experiments are reported that compare Neural Networks with a recently introduced variant known as Morphological Networks. A novel network node is then presented that is particularly suited to the problem of pattern recognition in multi-spectral data sets. More specifically, the node can exploit both spectral and spatial information, and implements both feature extraction and classification components of a typical image processing pipeline. Once trained, the network can be applied to large image data sets, or at the sensor to extract features of interest with two orders of magnitude speed-up compared to software implementations.

1. Introduction

Evolutionary Algorithms (EA) are a simple but powerful approach to optimization. The fundamental problem with EA is computation time. Field Programmable Gate Array (FPGA) implementations can provide significant speed-up compared to software implementations, and the training time of EA can be equal to, or less than conventional optimization techniques. This is a primary motivation of this paper. The combination of EA and FPGA is also seen widely in the field of Evolvable Hardware, where slightly different motivations are observed. EHW is often concerned with hardware design itself, and therefore EA are usually applied to more primitive hardware building blocks [1]. In practice, this distinction is often blurred. In the first case, accelerating software EA experiments often requires algorithmic trade-offs in order to implement chromosomes efficiently on FPGAs. In the second case, knowledge of high-level algorithms is often required to apply EHW design principles to practical problems.

This paper uses the combination of EA and FPGA to find hardware efficient solutions to pattern recognition problems. In this paper, the data sets considered are remotely sensed, multi-spectral imagery and the problem is referred to as Automatic Feature Extraction (AFE). AFE attempts to find algorithms that will consistently separate a feature of interest from the background in the presence of noise and uncertain conditions. The design space for hardware efficient AFE comes from two sources: algorithmic components of conventional AFE solutions and hardware resources available in FPGA devices. Network architectures have several properties that lead to efficient hardware implementation. These include:

- Inherent Parallel Processing: The final output of a network is a result of partial calculations performed by each node.
- Simple Processing Elements: Each node of the network need only be capable of solving part of a particular problem and therefore are relatively simple.
- Modular: Nodes are usually homogeneous across the network leading to simple large-scale designs.

For these reasons, networks appear to be a good starting point from which to develop hardware efficient AFE algorithms. This is not a new thing, and is partly why Neural Networks have received considerable attention for solving the classification aspect of AFE problems [2]. However, applying classification directly often leads to poor performance on out-of-sample data and therefore preprocessing and feature extraction are usually required. A good set of features will make classification easier and hopefully lead to good generalization.

One approach to this problem is the co-optimization of feature extraction and classification components. This means feature selection can be directed towards easily classifiable subsets, while simultaneously leading to simpler classifiers. An example of this approach was presented in [3] where a morphological shared weight neural network was used for an automatic target recognition problem. A problem with the co-optimization

0-7695-1180-5/01 $10.00 © 2001 IEEE

approach is that learning algorithms become complex. There are a large number of potentially useful transformations that could be used for feature extraction and optimization soon becomes intractable. This problem can be avoided by using EA.

In Section 2, an FPGA fitness evaluator is presented. It is implemented on the Firebird Custom Computer (CC) by Annapolis Microsystems [4]. This CC is based around a Virtex 2000E FPGA from Xilinx. It has four 64-bit local memories and a 5th 32-bit local memory and communicates with a host computer through a 64-bit PCI bus. The Firebird CC is used throughout the paper to accelerate evolution of network architectures.

Preliminary work is reported in Section 3 where a traditional two-layer Neural Network is compared to a more recently introduced Morphological Network [5] of comparable size. These networks are applied to a range of feature identification problems in multi-spectral imagery in Section 4. Section 5 builds on Section 3 and presents a novel network node that combines spectral classification techniques with spatial enhancement and feature extraction algorithms, in a self contained, modular design. This means *hybrid* feature extraction/ classification architectures that are scalable, inherently parallel and easily implemented. Section 6, makes an assessment of the approach by evolving a 3-layer multi-spectral network of these nodes known as POOKA.

2. Network Fitness Evaluation using CC

Figure 1 illustrates the major components for the CC fitness evaluator. The host/CC communication for a pipelined fitness evaluator is very efficient. Large volume training data is loaded to the CC local memory once, at the start of the run. During evolution, the host only writes to and reads on-chip registers. Large volume result data is retrieved once, at the end of the run.

The *Generalized Chromosome* implements the search space. It is configured with the on-chip *Config Registers*. By writing to these registers, a particular chromosome is configured which can then be evaluated. The *Generalized Chromosome* receives training data from one memory, performs the particular processing dictated by the configuration registers and then outputs the result to a second memory.

At the same time the truth data, also loaded to local memory, is passed to a delay unit. This is labeled *twdelay* in Figure 1 and implements latency equal to the *Generalized Chromosome* Pipeline. The latency-adjusted truth is then compared to the chromosome output in the *Fitness-Metric* unit. Input to the *Generalized Chromosome* is assumed to be signed 8-bit integers in the range {-127:127}. The fitness metric applies a threshold at 0. A binary metric can then be used which is a hamming distance weighted by the number of training points in the True and False classes.

$$Fitness = \left({}^{T_C}\!/_{T_T} \right) * 500 + \left({}^{F_C}\!/_{F_T} \right) * 500 \quad \textbf{(1)}$$

T_C is the number of true pixels correctly classified by the network and T_T is the total number of true pixels in the training set. Similarly, F_C is the total number of false pixels correctly classified and F_T is the total number of false pixels in the training set. A perfect classification will result in a score of 1000. Since finding all feature pixels is equivalent to finding all non-feature pixels, two fitness scores are calculated. The host program chooses the larger fitness value and assigns it to the chromosome. The Hamming metric is suitable only for two-class classification problems and only classification error is considered. No measure is made of the certainty in decision such as a distance from the decision boundary. The benefit of the weighted hamming metric is its simplicity of implementation in CC.

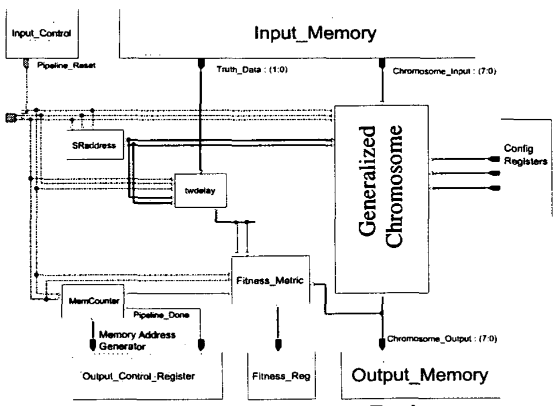
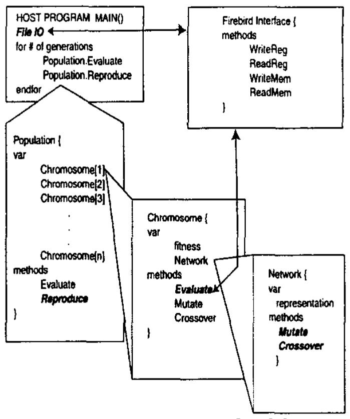


Figure 1: Pipeline Fitness Evaluator


```
HOST PROGRAM  MAIN()
File IO
for # of generations
        Population.Evaluate
        Population.Reproduce
endfor

Firebird Interface {
methods
        WriteReg
        ReadReg
        WriteMem
        ReadMem
}

Population {
var
        Chromosome[1]
        Chromosome[2]
        Chromosome[3]
        .
        .
        .
        Chromosome[n]
methods
        Evaluate
        Reproduce
}

Chromosome {
var
        fitness
        Network
methods
        Evaluate
        Mutate
        Crossover
}

Network {
var
  representation
methods
  Mutate
  Crossover
  }
```

Figure 2: Host Program Architecture

The overview of the host program is illustrated in Figure 2. This shows where the major components (bold) are implemented. The *Chromosome* object interfaces the software chromosome, encoded in the *Network* object, with the Firebird. For each chromosome, it sets the *Pipeline-Reset* and writes the configuration registers according to the representation stored in the Network object. It then clears the *Pipeline-Reset* and waits for the *Pipeline-Done* signal. Once received, the chromosome object retrieves the fitness score.

3. Morphological and Linear Perceptrons

Neural Networks have been implemented on CC by several researchers to accelerate both training and application. A fundamental operation in neural networks is multiplication. This can be expensive to implement on Field Programmable Gate Arrays (FPGA) as the number of nodes and connectivity within the network grows. Several techniques have been used to reduce this problem: implementation of partially connected neural networks [6], and time multiplexing of network nodes using run time reconfiguration [7].

Morphological Networks have much in common with Neural Networks but represent a fundamentally different approach. They have been shown to have equivalent classification power to neural networks [8] and can be implemented on FPGAs much more efficiently than neural networks. Traditional neural networks, using linear perceptrons, multiply the inputs by weights, and then sum the result. This linear operation is then followed by a non-linear thresholding operation to produce the perceptron output. This is illustrated in Figure 3a. In the morphological case, the operations of multiplication and addition are replaced by addition and maximum respectively. The morphological perceptron is illustrated in Figure 3b.

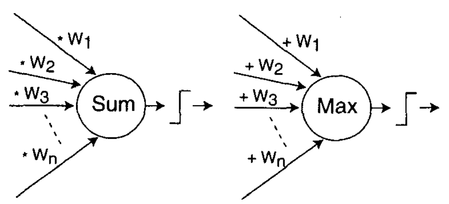


Figure 3: a) Linear and b) Morphological Perceptrons

The definition of the morphological perceptron comes from work presented in [5]. A multiplicative weight of ±1 is also associated with each input, which is described as an excitory/inhibitory weight. Several other researchers have suggested morphological networks, but in other forms [9], [10]. Both morphological and neural networks were implemented on the Firebird CC. Each perceptron had 12 inputs. Two-layer networks were built by combining four perceptrons for both the morphological and linear case. The binary outputs from both morphological and linear perceptrons were combined with a logical AND in the output node. This represents a 12-input, 4 hidden-layer, 1-output node network. Table 1 summarizes the resource requirements for the morphological and neural network implementations. There is a resource overhead when using the Firebird CC for memory, clock and PCI bus interface circuits. An 'empty' design was therefore also implemented so that the relative cost of the network architectures would be clear.

Table 1: Resource Comparison

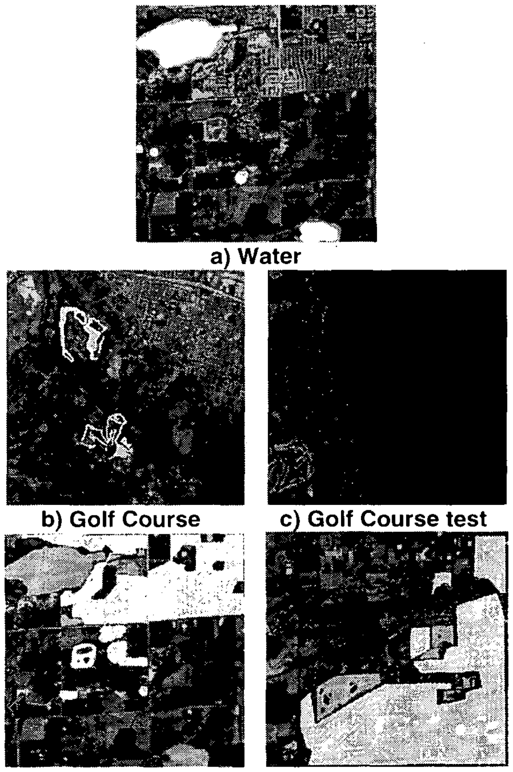
Design	Virtex Resources (slices)	Total (%)	Design (%)
Firebird Infrastructure	1935	10	0
Morphological Network	2432	12	2
Neural Network	3470	18	8

The neural network is approximately 4 times larger than the morphological design. A timing constraint of 66MHz was easily met by the morphological design with a pipeline latency of 2 clock cycles. The neural network design required 3 pipeline stages and several iterations of Place and Route were required. This suggests relative resource requirements may be greater for larger neural network implementations.

4. Application to Multi-Spectral Data Sets

In multi-spectral image processing, a D element vector in spectral space characterizes each pixel, where D is the number of spectral channels in the image. In most traditional approaches to multi-spectral AFE, classification is applied directly to this D-dimensional space [11]. Both networks were applied in this way to a 10-band multi-spectral data set. The training images that were used are depicted in Figure 4. The truth data, which defines the feature of interest, is depicted in the bright overlay. Non-feature is also specified, to allow some pixels to be don't care which do not contribute to fitness. The training data for non-feature is not shown.

The problem set was designed to span a range of difficulties. In Figure 4a, the feature of interest is water. This is the easiest problem of the three since water has a unique spectral signature. The second problem is to identify the golf courses. It is believed that this problem is of moderate difficulty but should have distinguishable spectral properties. The third training image specifies urban or 'built-up' areas as the feature of interest. Urban areas can include a wide variety of materials and therefore spectral signatures. It is believed this is the hardest problem to be solved with spectral information alone. Truth data was also specified for test images for the golf course and urban area problems so that a score could be obtained, and performance quantified.


a) Water

b) Golf Course c) Golf Course test

c) Urban d) Urban test

Figure 4: Multi-Spectral Problem Set

Both morphological and neural networks were evolved for 5 independent runs for each training image. The chromosome for the Neural Network is made up of multiplicative weights in the range {-7,7}. For the Morphological Network they are additive weights range {-128, 128: powers of 2} and the inhibitory/excitory weight {-1 or 1}. The search space size for the two networks is therefore approximately the same. A generational Genetic Algorithm with elitism was used [12].

For each run, a population of 200 networks was evolved for 500 generations. The results are summarized in Table 2 where a perfect classification will result in a score of 1000. For comparison, a software experiment was also implemented and a back-propagation learning algorithm was used. Multiple iterations of back-propagation were used, but were often caught local optima. There are many advanced learning algorithms that may find global optima more consistently, but this is also true for the EA. The best networks found in the 5 runs were then applied to the test problems, and results are also reported in Table 2.

Table 2: Summary of Results

Test Problem	**Morph. Network**		**Neural Network**		**Neural Back Prop.**
Training	*Mean*	*SD*	*Mean*	*SD*	
Water	999	0.1	999	0.08	999
Golf	954	3.73	962	1.65	937
Urban	759	9.26	788	7.69	756
Testing	*Best applied*		*Best applied*		
Golf	579		909		768
Urban	660		737		683

The Neural Network architecture outperformed the Morphological Network architecture in all problems. All architectures found the problems progressively more difficult as expected. The following conclusions are made:

1. If a network is to be implemented in hardware, evolutionary search is a well-motivated learning algorithm.
2. The Virtex FPGA can efficiently implement simple multi-bit arithmetic, multiplexing and small fixed-point multipliers.
3. Both spectral and spatial information are important in many feature identification problems of interest.

5. A Multi-Spectral Processing Node

The flexibility of EA suggests development of network architectures more suitable for multi-spectral image processing. Modern multi-spectral sensors are now being produced with high spatial resolution. To exploit these advances in sensor technology feature identification algorithms must utilize both spectral and spatial information. Figure 5 illustrates the spatial-spectral processing node.

The node has four inputs (four multi-spectral bands if applied directly to the data) and one output. The four inputs are first combined with a spectral processor that produces one output. This output is then input to a spatial processor. The two additional *Precision* components are used for controlling bit widths and handling precision.

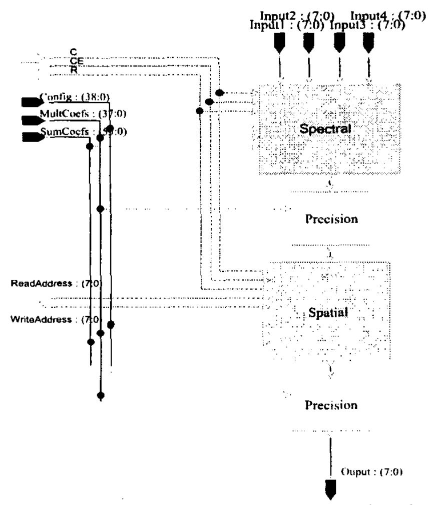


Figure 5: A Spatial-Spectral Processing Node

5.1 The spectral processor

The spectral processor can be considered a *hybrid* network node that incorporates both linear and morphological network functionality. Two coefficients are associated with each image plane and are applied according to Equation 2. The sum coefficients have a range between –127 and 127. The Multiplicative coefficients may assume values between –7 and 7.

$$Output = (Input + Sum_{Coef}) * Mult_{Coef} \quad \textbf{(2)}$$

Both images then pass to the *Arith-Morph-Mux* (AMM) unit that is illustrated in Figure 6. This processing block incorporates the fundamental flexibility of both spectral and spatial processor. The block is based around a programmable add/subtract unit. With the addition of control logic and multiplexers, the block can be configured to perform a number of functions of two inputs. A particular function is configured by setting 3 control lines *Mux, Func* and *Morph.* The corresponding functions are described in Table 3.

The spectral processor chromosome has four sets of coefficients and configuration bits for 3 Arith-Morph-Mux units (a two-layer binary tree).

5.2 The spatial processor

The Spatial Processor is applied to a single input image and implements functions of a 5 by 5 neighborhood. Figure 7 depicts the 5 by 5 register array associated with the operator. The input image is supplied to the processor through the row0 input. Four row outputs (on the right of the image) input to image-width row buffers whose outputs supply row1 through row4 inputs. Each register has an associated output so that the neighborhood function can be applied.

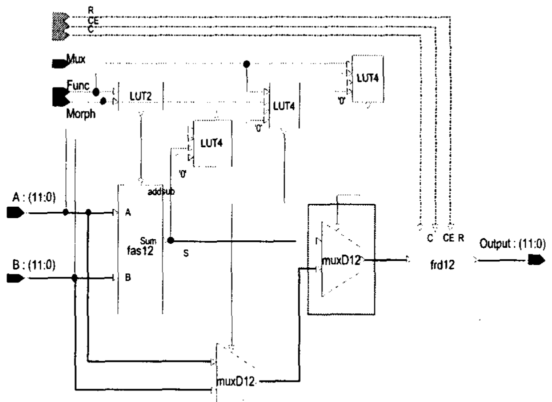


Figure 6: The Arith-Morph-Mux (AMM) unit

Table 3: Functions of the AMM unit

Control Bits (AMM)			Function Applied to pixels p_1 and p_2
Mux	Func	Morph	
0	0	0	Average $(p_1 + p_2)/2$
0	0	1	Difference $(p_1 - p_2)/2$
0	1	0	Maximum $\vee \{p_1, p_2\}$
0	1	1	Minimum $\wedge \{p_1, p_2\}$
1	*	0	Select p_1
1	*	1	Select p_2

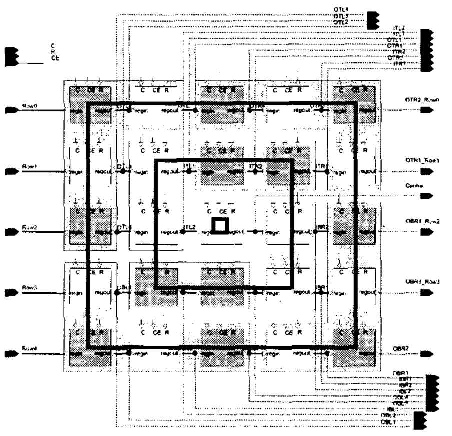


Figure 7: 5by5 Spatial Kernel

The spatial operator uses a particular order to combine the 25 input pixels into 1 output pixel. It is implemented with a binary tree made from *Arith-Morph-Mux* (AMM) units and *Arith-Morph-Abs* (AMA) units. The AMM units were discussed with respect to the Spectral Processor. The AMA unit replaces the multiplexing functionality with an absolute value operation. The modified configuration bits are summarized in Table 4.

Table 4: Configuration of AMA Unit

Control Bits (AMA)			Function
Mux	Func	Morph	Applied to pixels p_1 and p_2
1	*	0	$\lvert (p_1 + p_2)/2 \rvert$
1	*	1	$\lvert (p_1 - p_2)/2 \rvert$

Image-processing algorithms motivate the order of combination in the binary tree. At the top level, the 25 inputs are first combined into 3 *rings*. These are superimposed on Figure 5. The 5by5 ring has 16 inputs and the 3by3 ring has 8. The 3rd ring is the center pixel. These 3 values are combined with a spectral processor of 3 inputs. There are therefore both multiplicative and additive coefficients associated with each ring.

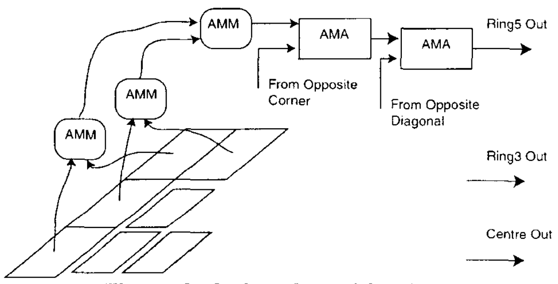


Figure 8: Order of combination (top-left corner of 5by5 mask)

The order in which pixels are combined in the ring is important. The order for the 5by5 case is illustrated in Figure 8. First, the corners of each ring are found. In the 5by5 case, a 4-input network of AMM units is used. This can be configured to find the average, maximum or minimum of the corner pixels (or any subset of). The corner averages can be used to estimate a gradient by combining opposite corners with the AMA unit. In this case, an absolute value of the difference represents the magnitude of an edge response [13]. Once opposite corners have been combined, the two diagonals that result, are combined with another AMA unit. This is most clearly seen in Figure 8.

Each ring can return an average, maximum, minimum or edge response. By associating weights with these rings, a *hybrid* linear/non-linear spatial filter is implemented. By setting weights appropriately, Gaussian smoothing and simple combinations of Gaussian functions can be implemented [14]. In this respect, the architecture is similar to that found in Convolutional Neural Networks [15]. However, in addition, the morphological aspect of the spatial filter means a rich variety of non-linear spatial filters can also be implemented. Examples of these include erosion, dilation and morphological range operators [16]. Aspects of the spatial processor worth particular note:

1. The combination of edge responses and smoothing is optimized by the EA. This is a powerful measure of texture. Convolution kernels by Laws [17] and their modifications in [18] are excellent examples of this type of linear filter.
2. The combination of linear and non-linear components is also optimized by the EA. For example, linear combinations of erosions and dilations. This is similar to hybrid L-filters [19] and pseudo-granulometries [20].

To encourage rotationally invariant operators, and to reduce the size of the search space, only one quarter of the tree is configured. The configuration for the top left quadrant of the tree is used in the other three quadrants. This is a common way of enforcing rotationally invariant structuring elements when optimizing morphological filters. Figure 9 illustrates the technique. Only the top-left portion of the neighborhood with gray background is configured. This configuration is then rotated through the four quadrants. In this example, a particular configuration produces a filter that depends only on pixels that are crossed. In terms of morphology this is known as a structuring element and the result can be seen on the right of Figure 9.

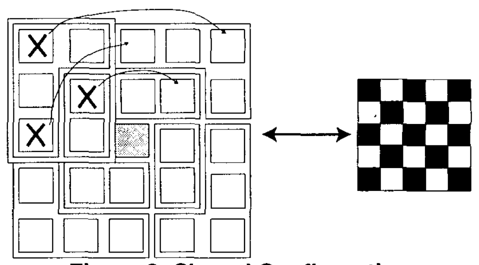


Figure 9: Shared Configuration

The spatial processor chromosome has three sets of coefficients associated with the 3 rings. With shared configuration, the AMM/AMA network requires 10 sets of the *mux*, *func* and *morph* configuration bits.

5.3 Band selection and precision

Input to the node is assumed to be 8 bit 2's complement values. Local memory resources of the Firebird CC dictated an upper limit of 12 channels. Each input to the node can receive input from any of the 12 channels. To accommodate a variable number of bands, tri-state bus resources were used to implement large multiplexers.

Input data is first scaled to the range {−127:127}. The multiplicative weights, used to combine inputs in the Spectral Processor, and the center, 3by3 and 5by5 rings in the Spatial Processors produce 12 bit signed outputs at full precision. It is desirable to maintain a consistent bit-width between input and output so that nodes can be easily cascaded to form networks. In this case, the EA is used to select which 8-bits should be used in the 12-bit output. This effectively scales the output, by dividing by powers of 2. Tri-state buses are used to multiplex the twelve bit input data. By setting control lines the bus effectively divides by 1,2,4,8 or 16. Data outside of the range {-127:127} after this scaling will saturate. This is a good example of how EA can be used to find solutions within a design space of finite hardware resources.

5.4 Node representation

The general structure of the node chromosome can be seen in Figure 10. The chromosome is made up of a combination of BITS, found in the AMM and AMA units, and an additional absolute value operation implemented in the precision unit. There are also integers, used in spectral/spatial processor coefficients, and in the precision unit divider.

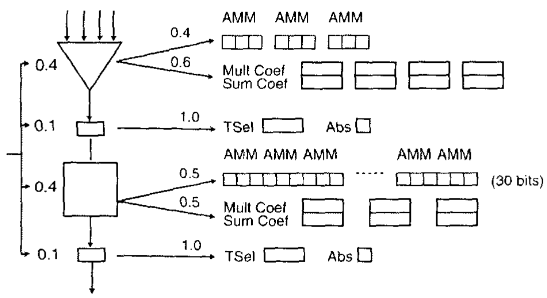


Figure 10: Node representation and mutation tree

Similar to the hardware, the chromosome is stored hierarchically in a number of objects in software. When mutation and crossover are applied to the node, there is a certain probability that they affect particular components. This can be considered a probability tree and Figure 10 shows the values used for mutation. Once mutation reaches a terminal node, there is equal probability of mutation within the subgroup. Crossover points are chosen in a similar way but are not discussed further.

6. POOKA: A Multi-Spectral Network

A 3-layer, 9 node network was implemented. The *Generalized Chromosome* for this network is illustrated in Figure 11. There are 16 inputs to the network at the first layer and therefore a total of 16 multi-spectral channels are chosen from the training data. This results in an additional 16 integers in the network chromosome. More than 1 node can get input from the same image channel. Each node in the second layer receives input from the four outputs of the first layer. The order of inputs for second layer nodes is kept constant for crossover. This means the first input of the 15 node is the same first input for 16, 17 and 18 nodes.

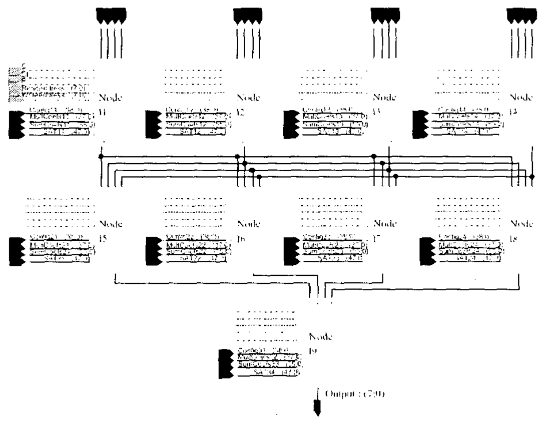


Figure 11: The 3-layer 9-node network

Several extensions to the Fitness Evaluator architecture of Section 2 were required to implement this larger network. These can only be briefly described in this paper. The output from each node is sent to local memory so that the host can retrieve it. Each node is also allocated a separate *twdelay* and *Fitness Metric* unit. This means fitness can be calculated on the output from each node. The training data associated with each node is also flexible. This means a total of 9 target classifications can be supplied to the network, one for each node.

Several extensions of the Host Program were also required. Multiple populations were used, each associated with a network node. Evolution within each population was kept independent to encourage specialization. This also means *incremental learning* techniques [21] can be implemented easily. Incremental evolution of the POOKA network can be considered a 3-stage process:

1. The four 1st layer nodes are evolved in parallel in four different populations. Since a fitness is calculated on the output from each node, these populations can be evolved independently.
2. In the second stage, the best 1st layer nodes in each population are configured and remain fixed. The 4 nodes in the 2nd layer are then evolved independently in 4 populations.
3. In the third stage, the best 2nd layer nodes are also configured. Both 1st and 2nd layer nodes remain fixed and the output node is evolved.

To maximally utilize the fitness evaluator resources, all 9 nodes should be involved in evolution at all times. This is not possible with the *Incremental Learning* approach. It is possible to evolve higher layers while lower-level nodes are evolved. This means the 1st, 2nd and 3rd layers are evolved in Stage 1. Only the 2nd and 3rd are evolved in Stage 2 and just the 3rd layer in Stage 3. This is the approach used in this paper, and is illustrated in Figure 12.

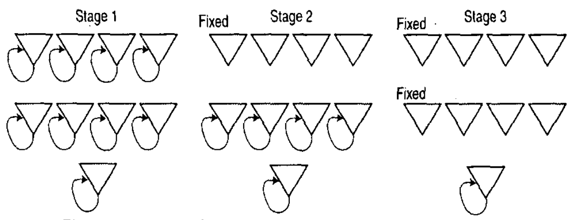


Figure 12: 3-Stage Incremental Learning

After *Incremental Evolution*, a variable number of *Optimization Cycles* are applied. This is most similar to *greedy* strategy suggested in [21] and is used to encourage collaboration between network nodes. This is an important concept in evolutionary neural networks and complex co-evolutionary strategies have been suggested [22]. In POOKA, optimization is a 9-stage process. The best nodes found after incremental evolution are configured. Each node is then taken in turn, and evolved for a number of generations with fitness calculated on the final output from the network (3rd layer output). This is illustrated in Figure 13. The Optimization Cycle is used to promote nodes that may not score well individually, but lead to better scores in the network as a whole.

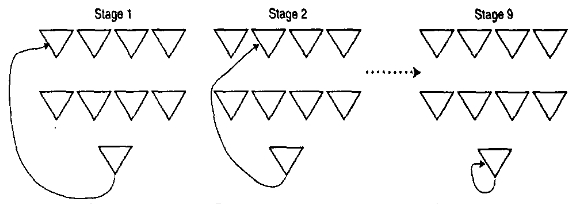


Figure 13: 9-Stage Optimization Cycle

6.1 Resource usage and evaluation of speed-up

The 9-node, 3-layer network was implemented at 50MHz and approximately 64% of the Virtex 2000E FPGA. Pre Place and Route the usage was estimated at 45%. This indicates significant room to optimize the design. All components of the network and fitness evaluator architectures were designed with structural VHDL to which placement constraints can be applied. This effectively allows the design to be manually placed, which can significantly improve density and clock rates. Evaluating speed-up of the architecture compared to software implementations is a difficult problem. Raw processing speed is not the only factor, since quality of the feature extraction algorithm is also important.

In the case of raw processing speed, one measure of performance is estimated by considering a high-level approximation of network components. In this case, the quality of algorithm is not considered, but rather the execution time of a particular chromosome. For the software, execution time was estimated by implementing a number of optimized image processing operators. For each Spectral Processor in the network, a linear combination was used. For each Spatial Processor, a 5by5 neighborhood average was calculated. The software experiment performed a total of 9 linear combinations of 4 images and 9 5by5 neighborhood averages. The execution times and relative speed-up are summarized in Table 5.

Table 5: Evaluation of Speed-up

Image Size (pixels)	Software Evaluation (Seconds)	RC Evaluation (Seconds)	Speedup
65536	0.18	0.0016	112
131072	0.36	0.0029	124
262144	0.71	0.0055	129
524288	1.39	0.0105	122
1048576	2.75	0.0201	136

6.2 Application to multi-spectral data sets

The network was trained on the 3 multi-spectral problems used in Section 4. The training time for each problem, including reading and writing training data was approximately 54.5 seconds. This included 52 seconds in evolution: 18.7 seconds of incremental development, followed by two optimization cycles of 16 seconds each. Populations of 200 nodes were evolved for approximately 180 generations.

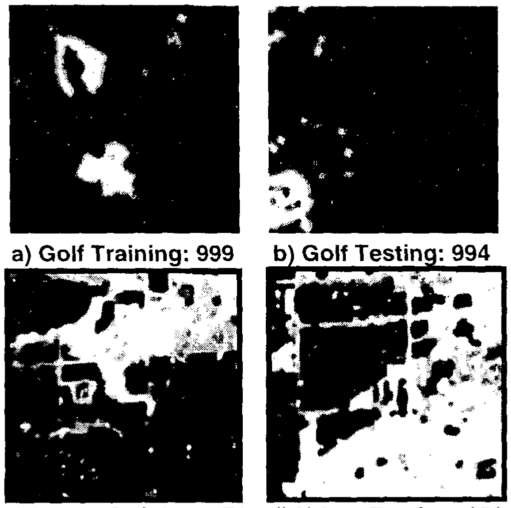


Figure 10: Output images from POOKA (fitness out of 1000)

Results for the water identification problem are not shown since the problem was easily solved and a perfect classification was obtained on the training data. Similar performance was observed for this problem in Section 4. The output images and fitness scores for the golf course and urban area problems are shown in Figure 10. Output images from the training data are shown on the left, and the output found on the test images are shown on the right. Fitness scores are calculated using Equation 1, and a perfect classification results in a score of 1000.

The POOKA network out-performed the morphological and neural networks of Section 4 on all problems. This is not surprising since these networks did not use spatial information. A more comprehensive comparison has been made to advanced spatio-spectral software techniques and results have been promising. This comparison will be presented in future publications.

7. Discussion and Future Directions

Network architectures are naturally suited to implementation on CC. Such architectures are easily scalable and inherently parallel. Evolutionary search is an effective means to optimize network architectures if the training time is reasonable. By implementing networks on CC, evolutionary search is a particularly attractive learning algorithm. The flexibility of evolutionary search means network architectures can be implemented with a particular problem in mind.

A novel network node was suggested for multi-spectral feature identification. It combines both spectral and spatial information using an image processing inspired, *morpho-linear* network. A 3-layer network of these nodes was described and preliminary results reported. Speed-up of two orders of magnitude compared to a software implementation of similar complexity was achieved. The effectiveness of evolutionary search, applied to large networks, can be improved by implementing multiple *Fitness Metric* units. This also has the advantage of allowing different target classifications to be associated with different nodes. Such flexibility may be used to direct evolution for more difficult problems. For example, a beach finder may be formed by combination of other high-level features such as land and water. Multiple *Fitness Metric* units allow some nodes to be trained to find water, and others to find beach, in parallel.

An interesting future direction will be to extend the network architecture presented by introducing *state*. This can be readily implemented with the Fitness Evaluator architecture since the output from each node is stored in the local memory. These output images can be considered the current state of the node. They can be fed back, as input to the node, to implement functions of state. Such architectures have several interpretations. In one sense, the network implements multi-layered gray-scale cellular automata [23], in which the transition rules are an engineered subset drawn from image processing. Another interpretation is a multi-layer cellular neural network [24], or more accurately, a cellular *morpho-linear* network. Fitness evaluation for these types of architectures will require multiple passes of the training data for each chromosome. These architectures are potentially applicable to real-time multi-spectral image processing.

8. References

1. Miller, J.F., D. Job, and V.K. Vassilev, *Principles in the Evolutionary Design of Digital Circuits - Part I.* Genetic Programming and Evolvable Machines, 2000. **1**(1): p. 7-35.
2. Bishop, C.M., *Neural Networks for Pattern Recognition.* 1995, Oxford: Oxford University Press.
3. Won, Y. and P.D. Gader. *Morphological Shared-Weight Neural Network for Pattern Classification and Automatic Target Detection.* in *IEEE International Conferenec on Neural Networks.* 1995.
4. Annapolis, M., *http://www.annapmicro.com/.* 2001.
5. Ritter, G.X. and P. Sussner. *An introduction to morphological neural networks.* in *13th International Conference on Pattern Recognition.* 1996. Vienna, Austria.
6. Chung, Y.Y., et al. *Implementing Neural Network in Custom Computers.* in *IEEE International Conference on Systems, Man and Cybernetics : Conference Theme : Intelligent Systems for Humans in a Cyberworld.* 1998. San Diego, California, USA: IEEE.
7. Eldredge, J.G. and B.L.Hutchings, *Run-Time Reconfiguration: a method for enhancing the functional density of SRAM-based FPGAs.* Journal of VLSI Signal Processing, 1996. **12**(1): p. 67-86.
8. Sussner, P. *Morphological Perceptron Learning.* in *Joint Conference on the Science and Technology of Intelligent Systems.* 1998. Maryland: IEEE.
9. Wilson, S.S. *Morphological Networks.* in *Visual Communications and Image Processing IV.* 1989: SPIE.
10. Yang, P. and P. Maragos, *Min-Max Classifiers: Learnability, Design and Application.* Pattern Recognition, 1995. **28**(6): p. 879-899.
11. Figueiredo, M.A. and C. Gloster. *Implementation of a Probabilistic Neural Network for Multi-spectral Image Classification on an FPGA Based Custom Computing Machine.* in *Vth Brazilian Symposium on Neural Networks.* 1998. Belo Horizonte, Brazil: IEEE Computer Society.
12. Mitchell, M., J.P. Crutchfield, and R. Das. *Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work.* in *First International Conference on Evolutionary Computation and Its Applications (EvCA'96).* 1996. Moscow, Russia.
13. Jain, A.K., *Fundamentals of Digital Image Processing.* Pretice Hall Information and System Sciences Series. 1989, New Jersey: Pretice Hall.
14. Macleod, I.D. and A. Rosenfeld, *The visibility of gratings: Spatial frequency channels or bar detecting units.* Vision Research, 1974. **14**: p. 909-916.
15. LeCun, Y. and B. Boser, *Convolutional networks for images, speech and time series,* in *The Handbook of Brain Science and Neural Networks,* M. Arbib, Editor. 1995, MIT Press: Cambridge, MA. p. 255-258.
16. Woods, R.C.G.a.R.E., *Digital Image Processing.* 1993, Reading, Massachusetts: Addison-Wesley Publishing Company.
17. Laws, K.I. *Texture energy measures.* in *Proceedings of Image Understanding Workshop.* 1979.
18. Pietikainen, M., A. Rosenfeld, and L.S. Davis, *Experiments with Texture Classification Using Averages of Local Pattern Matches.* IEEE Transactions on Systems, Man and Cybernetics, 1983. **SMC-13**(3).
19. Bovik, A.C., T. Huang, and D. Munson, *A generalization of median filtering using linear combinations or order statistics.* IEEE Trans. Acoust., Speech, Signal Processing, 1983. **31**: p. 1342-1350.
20. Aubert, A., D. Jeulin, and R. Hashimoto. *Surface Texture Classification from Morphological Transformations.* in *5th International Symposium on Mathematical Morphology.* 2000. Palo Alto, California: Kluwer Academic Publishers.
21. Potter, M.A. and K.A.D. Jong. *Evolving Neural Networks with Collaborative Species.* in *Proceedings of the 1995 Summer Computer Simulation Conference.* 1995. Ontario, Canada.
22. Moriarty, D.E. and R. Miikkulaiinen, *Forming Neural Networks through Efficient and Adaptive Coevolution.* Evolutionary Computation, 1998. **5**(4).
23. Sahota, P., M.F. Daemi, and D.G. Elliman. *Using Genetically Evolving Multi-Layer Cellular Automata for Image Processing.* in *Third Golden West Ineternational Conference on Intelligent Systems.* 1995. Netherlands: Kluwer Academis Publishers.
24. Chua, L.O., *CNN: A Paradigm for Complexity.* 1998: World Scientific Publishing Company.